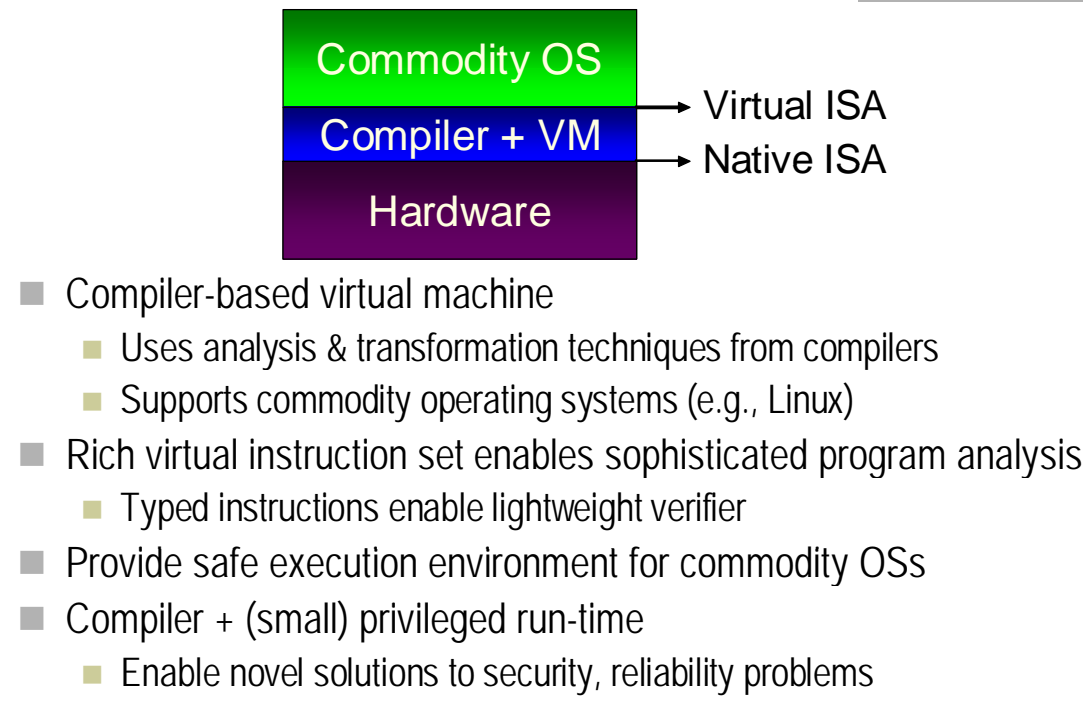


Secure Virtual Architecture

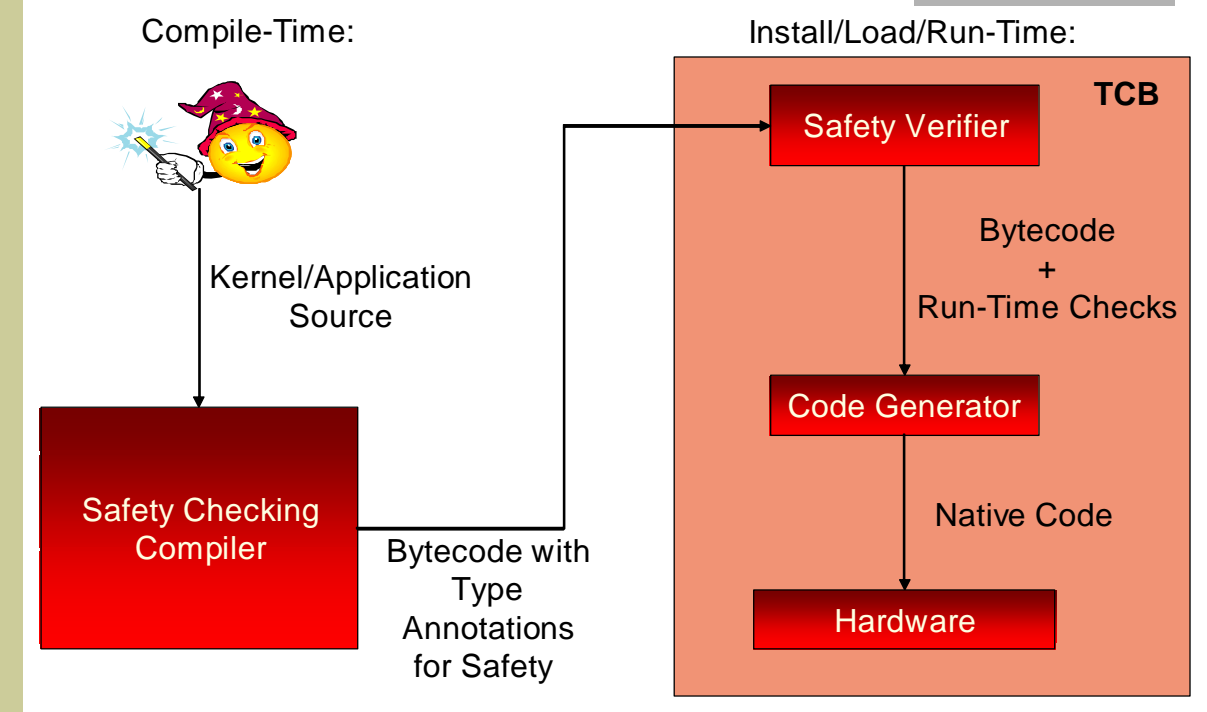


SVA Safety Guarantees

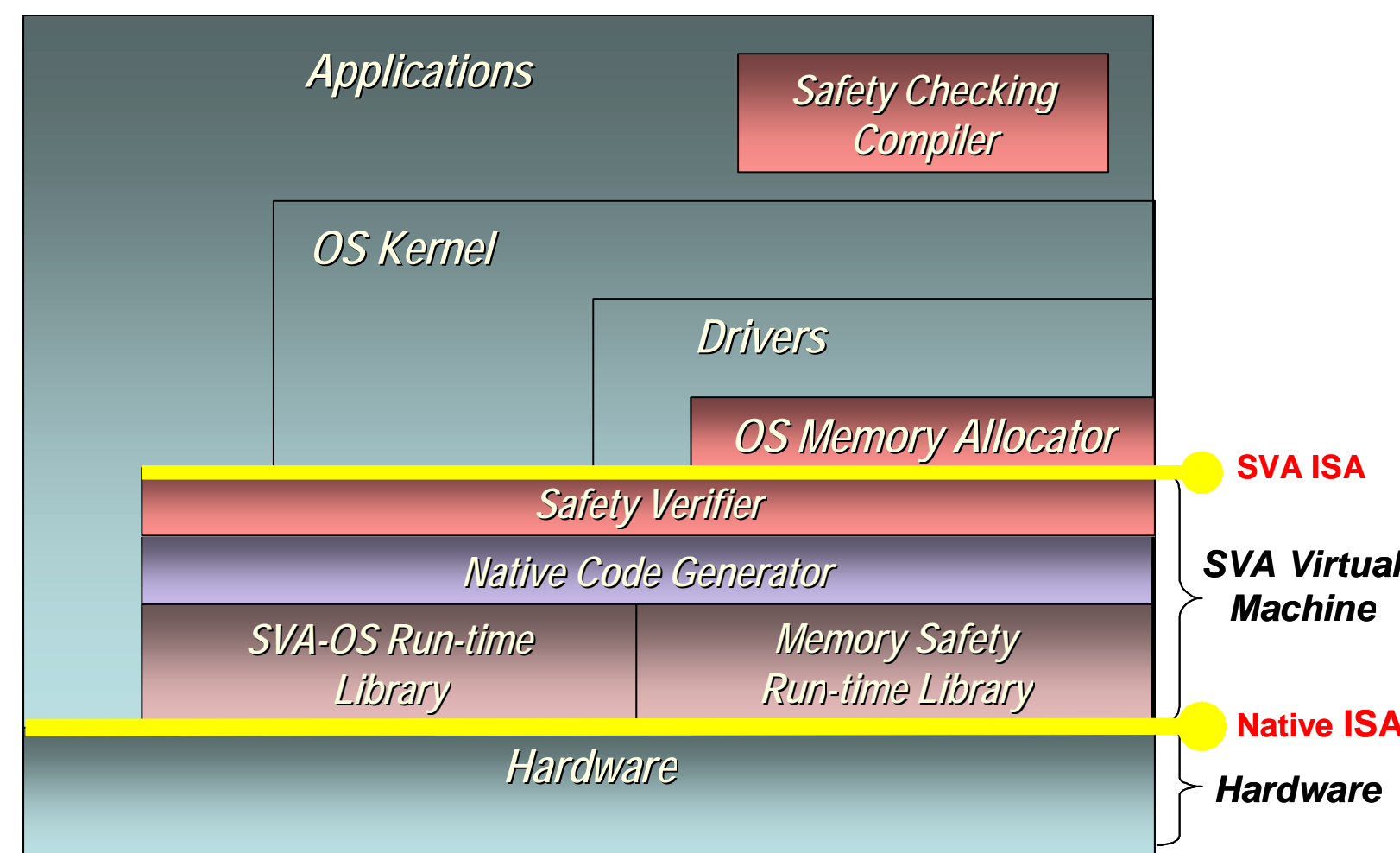
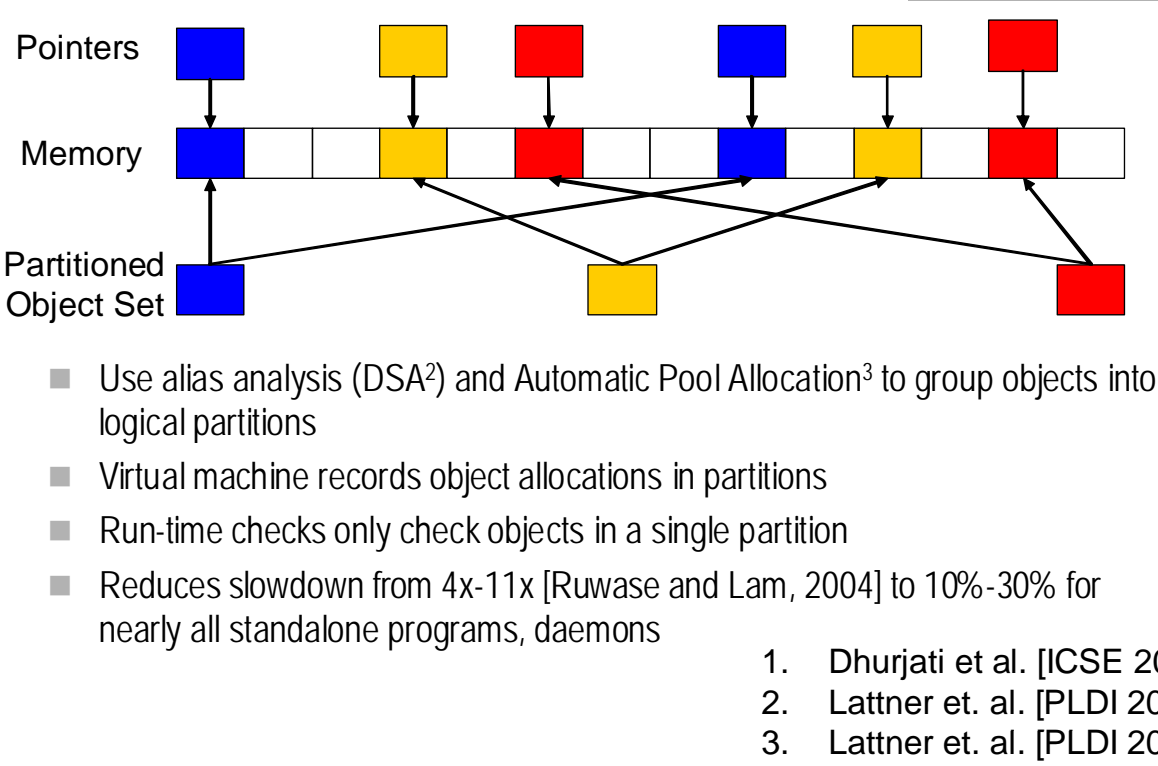
Safe Language	Secure Virtual Architecture
Array indexing within bounds	Array indexing within bounds
No uses of uninitialized variables	No uses of uninitialized variables
Type safety for all objects	Type safety for subset of objects
No uses of dangling pointers	Dangling pointers are harmless
Control flow integrity	Control flow integrity
Sound operational semantics	Sound operational semantics

- Dangling pointers & non-type-safe objects do *not* compromise other guarantees
- Novel bounds checking is compatible with external components
- Similar safety guarantees for applications, kernel

Software Flow



Improved Object Lookups¹



Virtual Instruction Set

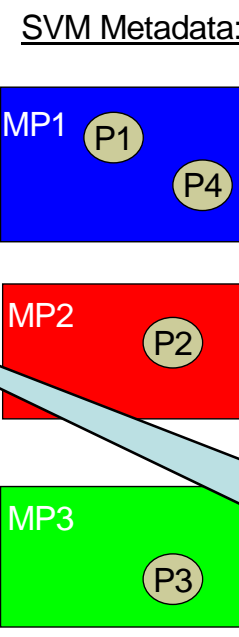
- SVA-Core
 - Based on LLVM^{1,2}
 - Typed, Explicit Control Flow Graph, Explicit SSA form
 - Sophisticated compiler analysis and transformation
- SVA-OS³
 - OS-neutral instructions support commodity OSs
 - Removes difficult to analyze assembly code
 - Encapsulates privileged operations
 - Like porting to a new hardware architecture

- [CGO 2004]
- <http://llvm.org>
- [WIOSCA 2006]

SVM Memory Safety

Insert object registration after allocation

```
Kernel Code:
P1=kmem_cache_alloc(inode_cache);
pchk_reg_obj (MP1, P1, reg_size(inode_cache));
...
Dest = &P1[index];
bounds = pchk_get_bounds (MP1, P1);
pchk_check_bounds (P1, Dest, bounds);
P2=vmalloc(size1);
pchk_reg_obj (MP2, P2, size1);
P3=kmem_cache_alloc(file_cache);
pchk_reg_obj (MP3, P3, reg_size(file_cache));
P4=kmem_cache_alloc(inode_cache);
pchk_reg_obj (MP1, P4, reg_size(inode_cache));
```



P1 and P4 are registered in the same metapool (same kernel pool)

Insert bounds check when the compiler cannot prove it safe

Linux Kernel Modifications

Original = Linux 2.4.22 with arch/i386

Section	Original LOC	SVA-OS	Allocators	Analysis	Total Modified
Arch-indep Core	9,822	41	76	3	120
Net Drivers	399,872	12	0	6	18
Net Protocols	169,832	23	0	29	52
Core FS	18,499	78	0	19	97
Ext3	5,207	0	0	1	1
Total Indep	603,232	154	76	58	288
Arch-dep	29,237	4,777	0	1	4,778

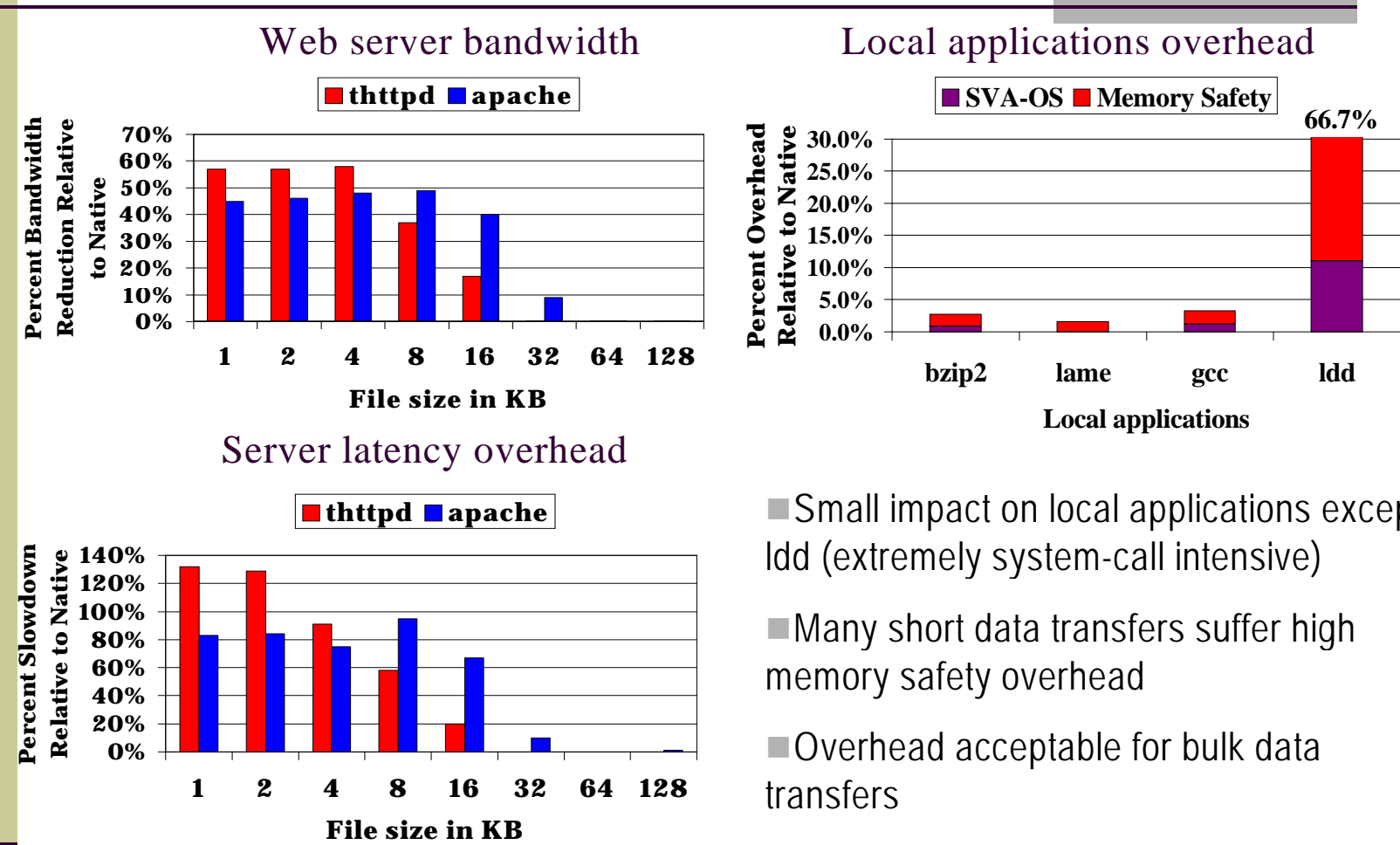
Majority of porting changes in architecture-dependent portion

Architecture-dependent code almost exclusively modified for SVA-OS port

Few memory allocator changes needed

Few changes to architecture-independent code

Performance



Exploits

- Tried 5 memory exploits that work on Linux 2.4.22
- Caught 4 out of 5 exploits
- Uncaught exploit due to code not instrumented with checks

BugTraq ID	Kernel Component	Caught?
11956	Console Driver	Yes!
10179	TCP/IP	Yes!
11917	TCP/IP	Yes!
12911	Bluetooth Protocol	Yes!
13589	ELF/Support Library	No

Future Work: Execution Environment

- Enhanced Memory Safety
 - Prevent MMU changes from violating safety guarantees
 - Prevent safety violations from errant I/O operations
 - Ensure state manipulation performed by OS is safe
- Cross-Boundary Analysis
 - Faulty OS cannot violate application memory safety
- Extend safety verifier to handle other security properties
 - Properties encodable in type system can be compact, efficient
 - More general proofs (e.g., as in Proof Carrying Code) possible

Future Work: Recovery

- Provide semantics & support to recover OS from unexpected errors
Don't convert memory safety errors into Denial-of-Service holes
- Errors may happen at arbitrary places in the kernel
 - Checks inserted for security policy enforcement
 - ECC errors on memory operations
 - Plain old bugs
 - Leverage existing error return semantics to provide recovery
 - OS kernels have many recovery paths
 - Develop techniques to transition from unexpected errors to existing error handlers
 - Preserve the kernel in a sane (i.e. correct) state when error occurs

Future Work: Novel Applications

- Trusted Logging
 - Virtual machine logs all OS operations securely
 - Compiler traces fine-grain operations within the kernel
 - Track changes to sensitive kernel/application data
 - Create causal event graphs to diagnose intrusions
- Enforce information flow policies uniformly within applications and kernel code
 - Enforce across application/kernel boundary
 - Enforce for type-unsafe code (C/C++)
 - Enforce for commodity OS kernel code (e.g., Linux, Mac OS X)
- Application Privacy
 - Hide private application memory from compromised OS
 - Automatically encrypt private data sent through OS for I/O

